(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 594 070 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
09.11.2005 Bulletin 2005/45

(51) Int Cl.[7]: **G06F 17/30**, G06F 11/14

(21) Application number: **05103690.3**

(22) Date of filing: **03.05.2005**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IS IT LI LT LU MC NL PL PT RO SE SI SK TR
Designated Extension States:
AL BA HR LV MK YU

(30) Priority: **03.05.2004 US 837932**
30.07.2004 US 903187

(71) Applicant: **MICROSOFT CORPORATION**
Redmond, Washington 98052-6399 (US)

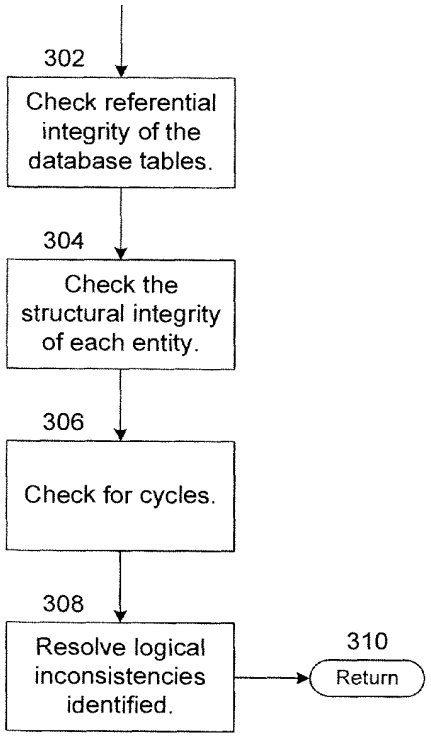
(72) Inventors:
- **Oks, Artem A.**
98052, Redmond (US)
- **Kodavalla, Hanumantha R.**
98052, Redmond (US)
- **Sleeman, Martin J.**
98052, Redmond (US)

(74) Representative: **Zimmer, Franz-Josef**
Grünecker, Kinkeldey,
Stockmair & Schwanhäusser
Anwaltssozietät
Maximilianstrasse 58
80538 München (DE)

(54) Consistency checking for a database management system

(57) The present invention is directed a logical consistency checker (LCC) working alone or in conjunction with a physical consistency checker (PCC) and/or a data reliability system (DRS) for a database files system of a hardware/software interface system. Logical data correction pertains to logical data corruptions for entities (e.g., items, extensions, and/or relationships in an item-based operating system, where an item-based operating system is one example of an item-based hardware/software interface system). In this regard, a LCC analyses and corrects logical damage to entities representatively stored in the data store in order to ensure that all such entities in said data store are both consistent and conform to the data model rules.

FIG. 3



EP 1 594 070 A2

Description

CROSS-REFERENCE

[0001] This application is a continuation-in-part of U. S. Patent Application No. 10/837,932 (Atty. Docket No. MSFT-3842), filed on May 3, 2004, entitled "SYSTEMS AND METHODS FOR AUTOMATIC DATABASE OR FILE SYSTEM MAINTENANCE AND REPAIR".

TECHNICAL FIELD

[0002] The present invention relates generally to file system management and, more particularly, to automatic file system maintenance and repair to ensure data reliability and consistency with regard to a data model. Various aspects of the present invention pertain to responding to and correcting logical data errors at a data entity level without losing other, down-level (child) data entities. In particular, various aspects of the present invention pertain specifically to the maintenance of logical data in an item-based hardware/software interface system.

BACKGROUND

[0003] While client database platforms (i.e., home and business desktop computers) use hardware of a quality that is much lower than on server platforms, even server-class hardware (controllers, drivers, disks, and so forth) can cause "physical" data corruption such that a read operation does not return what the database application wrote to the data store. Of course, this is clearly a more prolific problem with client database platforms (as opposed to server database platforms) for various reasons including without limitation the increased probability of a client machine been arbitrarily powered off in the midst of a write operation due to an unexpected power outage (which in turn leads to torn pages and potential database corruptions) whereas it is more common for server database systems to utilize uninterruptible power supplies to mitigate problems from power outages. Media decay is another source of "physical" data corruptions, where the physical storage media quite literally wears out over time. And yet another source of concern regarding reliability is the detection and recovery from "logical" corruptions caused by software errors whether inadvertent (e.g., bugs) or pernicious (e.g., viruses).

[0004] Traditionally maintenance and repair of a databases (and database file systems) has fallen to database managers and the like having a well-developed skill set and deep knowledge of database systems, or at least to individuals who are familiar with and regularly use database systems—by and large persons relatively skilled with regard to database technologies. On the other hand, typical consumer and business end-users of operating systems and application programs rarely work with databases and are largely ill-equipped to deal with database maintenance and repair issues.

[0005] While the disparate level of skill between these two groups has been largely irrelevant in the past, a database-implemented file system for a hardware/software interface system creates a scenario where these lesser-skilled end-users will be faced with database maintenance and repair issues they will largely be unable to resolve. Thus a business/consumer database-implemented operating system file system, or "database file system" (DBFS) for short, must be able to detect corruptions and recover its databases to a transactionally consistent state and, in the cases of unrecoverable data loss, the DBFS must then guarantee logical data consistency at the level atomic change units to said data are maintained (i.e., at the "item" level for an item-based DBFS). Moreover, for DBFSs running by default in a lazy commit mode, the durability of transactions committed just before an abnormal shutdown is not guaranteed and must be accounted for and corrected.

[0006] Moreover, while business/consumer end-users will greatly benefit from automating DBFS maintenance and recovery, database managers and those of greater database skills will also benefit from a technical solution for general database maintenance and repair. It is commonplace in the art for database administrators to utilize database tools (for example, the database tuning advisor provided with SQL Server 2000), but these tools do not directly address reliability but instead provide a means by which backups of the database are administered and managed—and not in a mostly-automated fashion, but instead requiring substantial database administrator involvement, particularly when database backups are not available or other repair issues arise. Thus an automated solution to address database reliability would also be beneficial for database administrators and other skilled database users.

[0007] A data reliability system (DRS) for a DBFS comprises a framework and a set of policies for performing database administration (DBA) tasks automatically and with little or no direct involvement by an end-user (and thus is essentially transparent to said end-user). For several embodiments, the DRS framework implements mechanisms for plugging error and event notifications, policies, and error/event handling algorithms into the DRS. More particularly, for these embodiments DRS is a background thread that is in charge of maintaining and repairing the DBFS in the background, and thus at the highest level the DRS guards and maintains the overall health of the DBFS. For certain embodiments, the DRS comprises the following features with regard to physical data corruption: (1) responding and correcting data corruptions at a page level for all page types; and (2) attempting a second level of recovery (rebuild or restore) for index page corruptions (clustered and non-clustered), data page corruptions, and page corruptions in the log file. Thus, for certain embodiments, the DRS comprising functionality for: (i) handling repair/restore data corruption cases; (ii) improving the

reliability and availability of the system; and (iii) keeping a DRS error/event history table for a skilled third party to troubleshoot database or storage engine problems if necessary.

[0008] While embodiments may address physical data corruption (i.e., correcting corrupted data in a database stored on the physical storage medium), a robust DRS should also address logical data corruptions to entities (e.g., items, extensions, and/or relationships) representatively stored in the data store in order to ensure that all such entities in said data store are both consistent and conform to the data model rules.

SUMMARY

[0009] Various embodiments of the present invention are directed a data reliability system (DRS) for a DBFS, said DBFS comprising a file system (logical data) maintained in a database (physical data) or, stated another way, comprising a database (physical data) that represents a file system (logical data). The DRS may comprise a framework and a set of policies for performing database administration (DBA) tasks automatically and with little or no direct involvement by an end-user (and thus is essentially transparent to said end-user). The DRS framework implements mechanisms for plugging error and event notifications, policies, and error/event handling algorithms into the DRS. More particularly, for these embodiments DRS is a background thread that is in charge of maintaining and repairing the DBFS in the background, and thus at the highest level the DRS guards and maintains the overall health of the DBFS.

[0010] For various embodiments of the present invention, the DRS comprises the following features:

- Physical Data Correction: responding to and correcting physical data corruptions at a page level for all page types, and which may include attempts to rebuild or restore operations for index page corruptions (clustered and non-clustered), data page corruptions, and page corruptions in the log file.
- Logical Data Correction: responding to and correcting logical data corruptions for "entities," e.g., items, extensions, and/or relationships in an item-based operating system (an item-based operating system being one example of an item-based hardware/software interface system).

[0011] In regard to the second bullet, several embodiments of the present invention are specifically directed to a logical consistency checker (LCC) that analyses and corrects logical "damage" to entities (e.g., items, extensions, and/or relationships) representatively stored in the data store in order to ensure that all such entities in said data store are both consistent and conform to the data model rules. For certain embodiments the LCC may be autonomous, while for other embodiments it may be coupled with a physical consistency checker (PCC) for detecting and correcting physical data corruptions, and/or for yet other embodiments the LCC may comprise a component of a DRS.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The foregoing summary, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0013] Fig. 1 is a block diagram representing a computer system in which aspects of the present invention may be incorporated;

[0014] Fig. 2 is a block diagram illustrating the structure of the data reliability system (DRS) in database file system (DBFS) representative of several embodiments of the present invention;

[0015] Fig. 3 is a process flow diagram illustrating an approach by which logically corrupted entities are ascertained for certain embodiments of the present invention;

[0016] Fig. 4 is a process flow diagram illustrating a three-prong approach for an LCC to resolve logical errors in an entity for certain embodiments of the present invention;

[0017] Figs. 5A and 5B are block diagrams that illustrate the replacement methodology regarding item entities for certain embodiments of the present invention; and

[0018] Figs. 6A and 6B are block diagrams that illustrate the replacement methodology regarding relationship entities for certain embodiments of the present invention.

DETAILED DESCRIPTION

[0019] The subject matter is described with specificity to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different steps or combinations of steps similar to the ones described in this document, in conjunction with other present or future technologies. Moreover, although the term "step" may be used herein to connote different elements of methods employed, the term should not be interpreted as implying any particular order among or between various steps herein disclosed unless and except when the order of individual steps is explicitly described.

[0020] The above summary provides an overview of the features of the invention. A detailed description of one embodiment of the invention follows. For various embodiments described below, the features of the

present invention are described as implemented in the MICROSOFT SQL SERVER database system (sometimes referred to herein simply as "SQL") alone or incorporated into the MICROSOFT WinFS file system for the next generation personal computer operating system (commonly referred to as "Windows Longhorn" or "Longhorn" for short). As mentioned above, SQL SERVER incorporates the MICROSOFT .NET Common Language Runtime (CLR) to enable managed code to be written and executed to operate on the data store of a SQL SERVER database. While the embodiment described below operates in this context, it is understood that the present invention is by no means limited to implementation in the SQL SERVER product. Rather, the present invention can be implemented in any database system that supports the execution of object-oriented programming code to operate on a database store, such as object oriented database systems and relational database systems with object relational extensions. Accordingly, it is understood that the present invention is not limited to the particular embodiment described below, but is intended to cover all modifications that are within the spirit and scope of the invention as defined by the appended claims.

Computer Environment

[0021] Numerous embodiments of the present invention may execute on a computer. Fig. 1 and the following discussion is intended to provide a brief general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand held devices, multi processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0022] As shown in Fig. 1, an exemplary general purpose computing system includes a conventional personal computer 20 or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start up, is stored in ROM 24. The personal computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like may also be used in the exemplary operating environment.

[0023] A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor 47, personal computers typically include other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

[0024] The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another per-

sonal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

[0025] When used in a LAN networking environment, the personal computer 20 is connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the personal computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0026] While it is envisioned that numerous embodiments of the present invention are particularly well-suited for computerized systems, nothing in this document is intended to limit the invention to such embodiments. On the contrary, as used herein the term "computer system" is intended to encompass any and all devices capable of storing and processing information and/or capable of using the stored information to control the behavior or execution of the device itself, regardless of whether such devices are electronic, mechanical, logical, or virtual in nature.

Overview of the Data Reliability System (DRS)

[0027] For several embodiments of the present invention, the data reliability system (DRS) is a thread that maintains and repairs the database in the background, and thereby guards the general health of the database file system (DBFS). Fig. 2 is a block diagram illustrating the structure of the DRS in the DBFS. In the figure, an operating system 202 providing operating system level services to a plurality of applications 232, 214, and 216, comprises a DBFS 222 logically coupled to a persistent data store 232. The operating system 202 further comprises a DRS 242 which is invoked 244 by the DBFS 222 whenever, for example, a page error 240 from among a plurality of pages 234, 236, and 238 in the persistent data store 232 is discovered, and the DRS 242 then performs repair operations in response to the page error 240.

[0028] Certain embodiments of the present invention provide that the DRS be extensible so that recovery policies and detection mechanisms may be updated after a DBFS has been released. Several embodiments are direct to a DRS that run repairs while the DBFS database is kept online. Still other embodiments are directed to run with full access to the DBFS store (that is, sysadmin privileges). Still other embodiments will have the ability to detect and react to failures in real time.

[0029] For several embodiments, DRS repairs will be transactional at the level change units to said data are maintained (i.e., at the "item" level for an item-based DBFS). For various embodiments, repairs will either completely recover an item or it will back out its changes (and thus never partially correct an error), and the DRS may also have the ability to continue the recovery/restoration work even if a reboot occurs half way thru the process. For several embodiments of the present invention, the DRS will subscribe to SQL events so that if SQL fires a general event, the DRS may intercept it and react (including without limitation 823/824 events). In addition, certain embodiments of the present invention are directed to a database engine that may be modified to send DRS-specific events for error conditions that the DRS is to specifically handle.

[0030] For various embodiments of the present invention, physical and/or logical corruptions will be detected whenever the DBFS reads or writes pages from disk, in which case SQL will then generate one of a host of errors depending on what type of corruption it is and will also fire specific DRS events to notify it of the specific error conditions, and the DRS will receive those errors and place them on in an incoming queue for processing.

[0031] For several embodiments of the present invention, ascertaining whether a page is physically corrupted may be accomplished by various means including, without limitation, (a) examining the checksum for a page and, if the checksum is invalid, the page is considered corrupt or (b) by examining the log serial number (LSN) to see if it is beyond the end of the log file (where an LSN is an integer that is incremented with each transaction so that if the last transaction in the log was LSN 432 and a page with a greater LSN is found then an out of order write error must have occurred.. In this regard, there are four major types of page corruptions that can effect the operation of a DBFS (in addition to other sources such as bugs, etc.), and these four types include torn pages, media decay, hardware failure, and out-of-order writes. Torn pages occur when a page of data is not correctly written atomically, and thus any part of the page may be corrupted because during a write only some of the sectors of a page make it to disk before the failure event, for example, a power failure or a sector write failure. Media decay occurs when a data pages bits have been corrupted by physical media decay. A hardware failure could arise for a variety of reasons related to the bus, the controller, or the hard disk device. As for out-of-order write, these errors stem from the fact that IDE drives cannot guarantee the order of writes to the disk, especially the IDE drive has write-caching enabled (turned on), and thus it is possible that writes to

the data store may occur out of order. If a partial series of out of order writes occur but are interrupted by a power failure, for example, then several errors may occur, such as the data page being written to disk before the associated log entry being written for example. While out-of-order errors can be detected by checking the log sequence numbers (LSN) on data pages, there is no easy way to do this short of reading every page.

Logical Consistency Checker

[0032] Various embodiments of the present invention are specifically directed to a logical consistency checker (LCC) that analyses and corrects logical "damage" to entities (e.g., items, extensions, and/or relationships) representatively stored in the data store in order to ensure that all such entities in said data store are both consistent and conform to the data model rules. For certain embodiments the LCC may be autonomous, while for other embodiments it may be coupled with a physical consistency checker (PCC) for detecting and correcting physical data corruptions, and/or for yet other embodiments the LCC may comprise a component of a DRS.

[0033] For a file system built with database technology (a database file system), logical consistency is distinct and separate from physical consistency in the sense that the latter (physical consistency) refers to the database structure itself and the physical storage of that database on a storage medium whereas the former (logical consistency) refers to the logical data schema that is represented by the data stored in said database and represents the file system of the hardware/software interface system.

[0034] Although physical consistency is related to logical consistency in certain regards (as discussed herein below), certain embodiments of the present invention are primarily directed to ensuring logical consistency. Of course, physical damage resulting in physical inconsistency (e.g., a disk sector goes bad, said disk sector containing a portion of said database structure) may also result in damage to logical consistency (e.g., loss of data for an entity stored in said database at said bad disk sector), but not all logical damage necessarily corresponds to physical damage (e.g., a logical error resulting from a software bug that violates a data model rule). Consequently, logical inconsistencies can be divided into two types: (i) logical inconsistencies due to physical damage, and (ii) logical inconsistencies due to violations of at least one data model rule (for example, all entity property values must be within a rule-specified range, an entity must have all of its constituent parts, and item must have at least one holding relationship, and so on and so forth).

[0035] In general, "repairing" a logical error is inherently inferior to "restoring" the data in which the error occurs because a backup of that data is likely a good copy (or can be used to reconstruct a good copy) of the data that was damaged or lost. Therefore, restoration techniques are preferred to repair techniques.

[0036] For several embodiments of the present invention, ascertaining whether any entities on a page is logically corrupted may be accomplished using the approach illustrated in Fig. 3. For this approach, at step 302 the LCC checks the database tables for the referential integrity of entities existing in the DBFS (the "entity externals") and then, at step 304, the LCC checks the structural integrity of each entity (the "entity internals," e.g., constraints and relationships) to ensure no data model rule violations. (For certain embodiments, the LCC checks every entity in the database.) At step 306, the LCC for certain embodiments may also check for cycles (e.g., where entity A has a holding relationship to entity B and entity B has a holding relationship to entity A). After the aforementioned checks have been completed, the LCC then, at step 308, resolves the logical inconsistencies identified.

[0037] For several embodiments of the present invention, the LCC utilizes a three-prong approach to resolving logical errors as illustrated in Fig. 4. First, at step 402, the LCC will attempt to perform a page-level restoration (PLR) using the most recent snapshot of the page and the transaction log to perfectly restore the page having the error. However, if a PLR is not possible or cannot correct the error at step 404, the LCC will then, at step 406, attempt an entity-level restoration (ELR) by first determining the specific entity that is damaged and then restoring that entity from another source (such as a backup or a sync replica). If both a PLR and an ELR are not possible or cannot correct the error at 408, then at step 410 the LCC will replace the damaged entity in the store with a dummy entity (DE) in order to guarantee a consistent view of the file system store as discussed below.

[0038] By replacing a damaged entity with a DE, the LCC ensures that removal of said damaged entity does not corrupt children entities of said damaged entity—that is, the LCC prevents cascading corruptions down-level from the corrupted entity to its children. To accomplish this, the DE essentially replaces the damaged entity but retains as much information from the damaged entity as possible. If the damaged entity is an item, for example, the replacing DE will retain as much of the property data as it can, as well as all of the relationships to other items. On the other hand, if the damaged entity is a relationship, the replacing DE will continue to connect the items to which it pertains together. The damaged entity, meanwhile, is moved to (for items) or logged in (for relationships) a broken item folder (BIF). When the damaged entity is an item, the BIF will have a relationship (e.g., a holding relationship) with the damaged entity.

[0039] Figs. 5A and 5B are block diagrams that illustrate the replacement methodology regarding items for certain embodiments of the present invention. In Fig. 5A, which illustrates a set of items and relationships, I1 is a parent item, I2 is a child item of I1 via relationship

R12,13 is a child item of I2 via relationship I23, and I4 and I5 are children items of I3 via relationships R34 and R35 respectively. In this example, item I2 is corrupted by, for example, a buggy application and, as a result, item I2 is now in violation of the data model rules. In Fig. 5B, the LCC, having identified I2 as a corrupted item, first creates DE I2' and establishes a first relationship R12' between the DE I2' and its parent I1 and a second relationship R23' between the DE I2' and its child I3. For certain embodiments, the DE is given the same item identification number (ItemID) as the corrupted item. Corrupted item I2 is then moved to the BIF and with a holding relationship Rh between the BIF item and the damaged item I2.

[0040] For certain embodiments, the new relationships R12' and R23' may in fact be the original relationships R12 and R23 that are updated to associated with I2' instead of I2. For other embodiments, R12' and R23' may be entirely new relationships and, for certain such embodiments, R12 and R23 may be retained as dangling relationships with damaged item I2 in the BIF. Regardless, the DE effectively preserves the parent/child structure for the dataset and thereby prevents an error to I2 to cascade as errors in I3, I4, and I5 that might otherwise be unreachable from 11.

[0041] Figs. 6A and 6B are block diagrams that illustrate the replacement methodology regarding relationships for certain embodiments of the present invention. In Fig. 6A, which illustrates a partial set of items and relationships, 11 is a parent item, and 12 is a child item of 11 via relationship R12. In this example, relationship R12 is corrupted by, for example, a virus that, as a result, causes relationship R12 to have a property value, e.g., a security property value, outside of some predetermined permitted range in the data model. In Fig. 6B, the LCC, having identified R12 as a corrupted relationship, first creates DE R12' between the I2 and its parent I1, and corrupted relationship R12 is eliminated (and not moved to the BIF since a relationship cannot exist alone in the BIF) and item I1, which owned the relationship R12, is logged into the BIF (the BIF having a special log for this purpose and shown herein, for example, as a log item).

[0042] In regard to synchronization, and to avoid the possibility of a corrupted entity being erroneously synchronized from partner to partner (thereby spreading the corruption), certain embodiments of the present invention compartmentalize identified and/or corrupted corruptions by marking DEs with a special "non authoritative" flag (e.g., a singe bit) that effectively notifies any sync replica that has a good copy of this entity to overwrite this entity (at which point the non-authoritative bit is cleared). Similarly, if a DE is subsequently modified (such as by an end-user), certain embodiments will also mark the DE as "non-authoritative and modified" to ensure that a conflict resolution procedure is employed as between the modified DE and the good copy of the original item on a replica, and the non-authoritative and modified marking will be removed as soon at the conflict has been resolved.

Additional Functionality

[0043]

- As described herein, item extensions are considered part of the owning item, and thus any extension corruption is deemed item corruption. However, in certain alternative embodiments, extensions may be treated as distinct and separate from their owning items.
- For certain embodiments of the present invention, a LCC is run on entities following a restoration operation performed to correct physical corruption.
- For certain embodiments, a LCC will attempt to repair corrupted items first before attempting to correct corrupted relationships in order to avoid the detection of "phantom" corruptions that might result if an item is not corrected before the relationships that depend on it is corrected.
- For certain embodiments, the BIF is a folder item created may be created by the LCC if one does not already exist in the DBFS. This folder may hold any type of item in the DBFS (but not relationships for certain embodiments) and the folder may be located off of the root of the default database store (for easy access and locating).
- For certain embodiments, any items without backing files will be put into the BIF, as well as any files without corresponding items will also be placed in the BIF.
- For certain embodiments, when a damaged item is moved to the BIF, the BIF may also store information pertaining to why the damaged item was moved to the BIF.

Conclusion

[0044] The various system, methods, and techniques described herein may be implemented with hardware or software or, where appropriate, with a combination of both. Thus, the methods and apparatus of the present invention, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium, wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. In the case of program code execution on programmable computers, the computer will generally include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. One or more programs are preferably implemented in a high level procedural or ob-

ject oriented programming language to communicate with a computer system. However, the program(s) can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language, and combined with hardware implementations.

[0045] The methods and apparatus of the present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via any other form of transmission, wherein, when the program code is received and loaded into and executed by a machine, such as an EPROM, a gate array, a programmable logic device (PLD), a client computer, a video recorder or the like, the machine becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, the program code combines with the processor to provide a unique apparatus that operates to perform the indexing functionality of the present invention.

[0046] While the present invention has been described in connection with the preferred embodiments of the various figures, it is to be understood that other similar embodiments may be used or modifications and additions may be made to the described embodiment for performing the same function of the present invention without deviating there from. For example, while exemplary embodiments of the invention are described in the context of digital devices emulating the functionality of personal computers, one skilled in the art will recognize that the present invention is not limited to such digital devices, as described in the present application may apply to any number of existing or emerging computing devices or environments, such as a gaming console, handheld computer, portable computer, etc. whether wired or wireless, and may be applied to any number of such computing devices connected via a communications network, and interacting across the network. Furthermore, it should be emphasized that a variety of computer platforms, including handheld device operating systems and other application specific hardware/software interface systems, are herein contemplated, especially as the number of wireless networked devices continues to proliferate. Therefore, the present invention should not be limited to any single embodiment, but rather construed in breadth and scope in accordance with the appended claims.

Claims

1. A logical consistency checker (LCC) for a database file system (DBFS), said LCC comprising at least one system for:

   checking referential integrity of a set of tables in a database corresponding to said DBFS for at least one logical inconsistency; and

   checking structural integrity of a set of entities represented in said DBFS for at least one logical inconsistency.

2. The system of claim 1 further comprising at least one subsystem for checking for at least one cycle that would constitute a logical inconsistency.

3. The system of claim 1 further comprising at least one subsystem for resolving at least one logical inconsistency.

4. The system of claim 3 wherein said at least one subsystem for resolving at least one logical inconsistency attempts at least one solution from among the following groups of solutions: a page-level restoration, a entity-level restoration, and replacing a damaged entity with a dummy entity.

5. The system of claim 4 wherein said at least one subsystem for resolving at least one logical inconsistency first attempts a page-level restoration and, if not successful, then attempts an entity-level restoration and, if not successful, then replaces at least one damaged entity corresponding to said at least one logical inconsistency with a dummy entity.

6. The system of claim 4 wherein, if said entity is an item, said solution of replacing a damaged entity with a dummy entity comprises:

   creating said dummy entity and redirecting at least one relationship belonging to said damaged entity with said dummy entity; and
   moving said damaged entity to a broken item folder.

7. The system of claim 4 wherein, if said entity is a relationship, said solution of replacing a damaged entity with a dummy entity comprises:

   creating said dummy entity as a relationship corresponding to the relationship of the damaged item; and
   logging said damaged entity in a broken item folder.

8. An automated data reliability system (DRS) for a database file system (DBFS), said DRS comprising:

   a set of policies for performing database administration (DBA) tasks;
   a subsystem for resolving a set of physical data corruptions at a page level; and
   a subsystem for resolving a set of logical data corruptions at an entity level.

9. The system of claim 8 wherein said DBFS is a com-

ponent of an item-based hardware/software interface system.

10. The system of claim 8 further comprising an interface for adding, deleting, and modifying at least one functionality from among the following group of functionalities: error and event notifications, policies, and error/event handling algorithms.

11. The system of claim 8 wherein said DRS operates as a background thread.

12. The system of claim 8 wherein said subsystem for resolving a set of logical data corruptions at an entity level is executed after the execution of said subsystem for resolving a set of physical data corruptions at a page level.

13. The system of claim 8 wherein said subsystem for resolving a set of logical data corruptions at an entity level attempts to repair a corrupted item before attempting to correct a corrupted relationship corresponding to said corrupted item.

14. A method for a logical consistency checker (LCC) for a database file system (DBFS) to address logical inconsistencies on an entity-level, said method comprising:

checking referential integrity of a set of tables in a database corresponding to said DBFS for at least one logical inconsistency; and
checking structural integrity of a set of entities represented in said DBFS for at least one logical inconsistency.

15. The method of claim 14 further comprising checking for at least one cycle that would constitute a logical inconsistency.

16. The method of claim 14 further comprising resolving at least one logical inconsistency.

17. The method of claim 16 wherein said element of resolving at least one logical inconsistency comprises the attempt of at least one solution from among the following groups of solutions: a page-level restoration, a entity-level restoration, and replacing a damaged entity with a dummy entity.

18. The method of claim 17 wherein said element of resolving at least one logical inconsistency comprises:

attempting at a page-level restoration;
if said attempt at a page-level restoration is not successful, then attempting an entity-level restoration; and
if said attempt at an entity-level restoration is not successful, then replacing at least one damaged entity corresponding to said at least one logical inconsistency with a dummy entity.

19. The method of claim 17 wherein, if said entity is an item, said solution of replacing a damaged entity with a dummy entity further comprises:

creating said dummy entity and redirecting at least one relationship belonging to said damaged entity with said dummy entity; and
moving said damaged entity to a broken item folder.

20. The method of claim 17 wherein, if said entity is a relationship, said solution of replacing a damaged entity with a dummy entity comprises:

creating said dummy entity as a relationship corresponding to the relationship of the damaged item; and
logging said damaged entity in a broken item folder.

21. A method for an automated data reliability system (DRS) for a database file system (DBFS) to address logical inconsistencies on an entity-level, said method comprising:

utilizing a set of policies for performing database administration (DBA) tasks;
resolving a set of physical data corruptions at a page level; and
resolving a set of logical data corruptions at an entity level.

22. The method of claim 21 wherein said DBFS is a component of an item-based hardware/software interface system.

23. The method of claim 21 further comprising an interface for adding, deleting, and modifying at least one functionality from among the following group of functionalities: error and event notifications, policies, and error/event handling algorithms.

24. The method of claim 21 wherein said DRS operates as a background thread.

25. The method of claim 21 wherein said element of resolving a set of logical data corruptions at an entity level is executed after the execution of said subsystem for resolving a set of physical data corruptions at a page level.

26. The method of claim 21 wherein said element of resolving a set of logical data corruptions at an entity

level further comprises attempting to repair a corrupted item before attempting to correct a corrupted relationship corresponding to said corrupted item.

27. A computer-readable medium comprising computer-readable instructions for a logical consistency checker (LCC) for a database file system (DBFS) to address logical inconsistencies on an entity-level, said computer-readable instructions comprising instructions for:

checking referential integrity of a set of tables in a database corresponding to said DBFS for at least one logical inconsistency; and
checking structural integrity of a set of entities represented in said DBFS for at least one logical inconsistency.

28. The computer-readable instructions of claim 27 further comprising instructions for checking for at least one cycle that would constitute a logical inconsistency.

29. The computer-readable instructions of claim 27 further comprising instructions for resolving at least one logical inconsistency.

30. The computer-readable instructions of claim 29 further comprising instructions whereby said element of resolving at least one logical inconsistency comprises the attempt of at least one solution from among the following groups of solutions: a page-level restoration, a entity-level restoration, and replacing a damaged entity with a dummy entity.

31. The computer-readable instructions of claim 30 further comprising instructions whereby said element of resolving at least one logical inconsistency comprises:

attempting at a page-level restoration;
if said attempt at a page-level restoration is not successful, then attempting an entity-level restoration; and
if said attempt at an entity-level restoration is not successful, then replacing at least one damaged entity corresponding to said at least one logical inconsistency with a dummy entity.

32. The computer-readable instructions of claim 30 further comprising instructions whereby, if said entity is an item, said solution of replacing a damaged entity with a dummy entity further comprises:

creating said dummy entity and redirecting at least one relationship belonging to said damaged entity with said dummy entity; and
moving said damaged entity to a broken item folder.

33. The computer-readable instructions of claim 30 further comprising instructions whereby, if said entity is a relationship, said solution of replacing a damaged entity with a dummy entity comprises:

creating said dummy entity as a relationship corresponding to the relationship of the damaged item; and
logging said damaged entity in a broken item folder.

34. A computer-readable medium comprising computer-readable instructions for an automated data reliability system (DRS) for a database file system (DBFS) to address logical inconsistencies on an entity-level, said computer-readable instructions comprising instructions for:

utilizing a set of policies for performing database administration (DBA) tasks;
resolving a set of physical data corruptions at a page level; and
resolving a set of logical data corruptions at an entity level.

35. The computer-readable instructions of claim 34 further comprising instructions whereby said DBFS is a component of an item-based hardware/software interface system.

36. The computer-readable instructions of claim 34 further comprising instructions for an interface for adding, deleting, and modifying at least one functionality from among the following group of functionalities: error and event notifications, policies, and error/event handling algorithms.

37. The computer-readable instructions of claim 34 further comprising instructions whereby said DRS operates as a background thread.

38. The computer-readable instructions of claim 34 further comprising instructions whereby said element of resolving a set of logical data corruptions at an entity level is executed after the execution of said subsystem for resolving a set of physical data corruptions at a page level.

39. The computer-readable instructions of claim 34 further comprising instructions whereby said element of resolving a set of logical data corruptions at an entity level further comprises attempting to repair a corrupted item before attempting to correct a corrupted relationship corresponding to said corrupted item.

40. A hardware control device for an automated data reliability system (DRS) for a database file system (DBFS) to address logical inconsistencies on an entity-level, said device comprising means for:

utilizing a set of policies for performing database administration (DBA) tasks;
resolving a set of physical data corruptions at a page level; and
resolving a set of logical data corruptions at an entity level by:

checking referential integrity of a set of tables in a database corresponding to said DBFS for at least one logical inconsistency, and
checking structural integrity of a set of entities represented in said DBFS for at least one logical inconsistency.

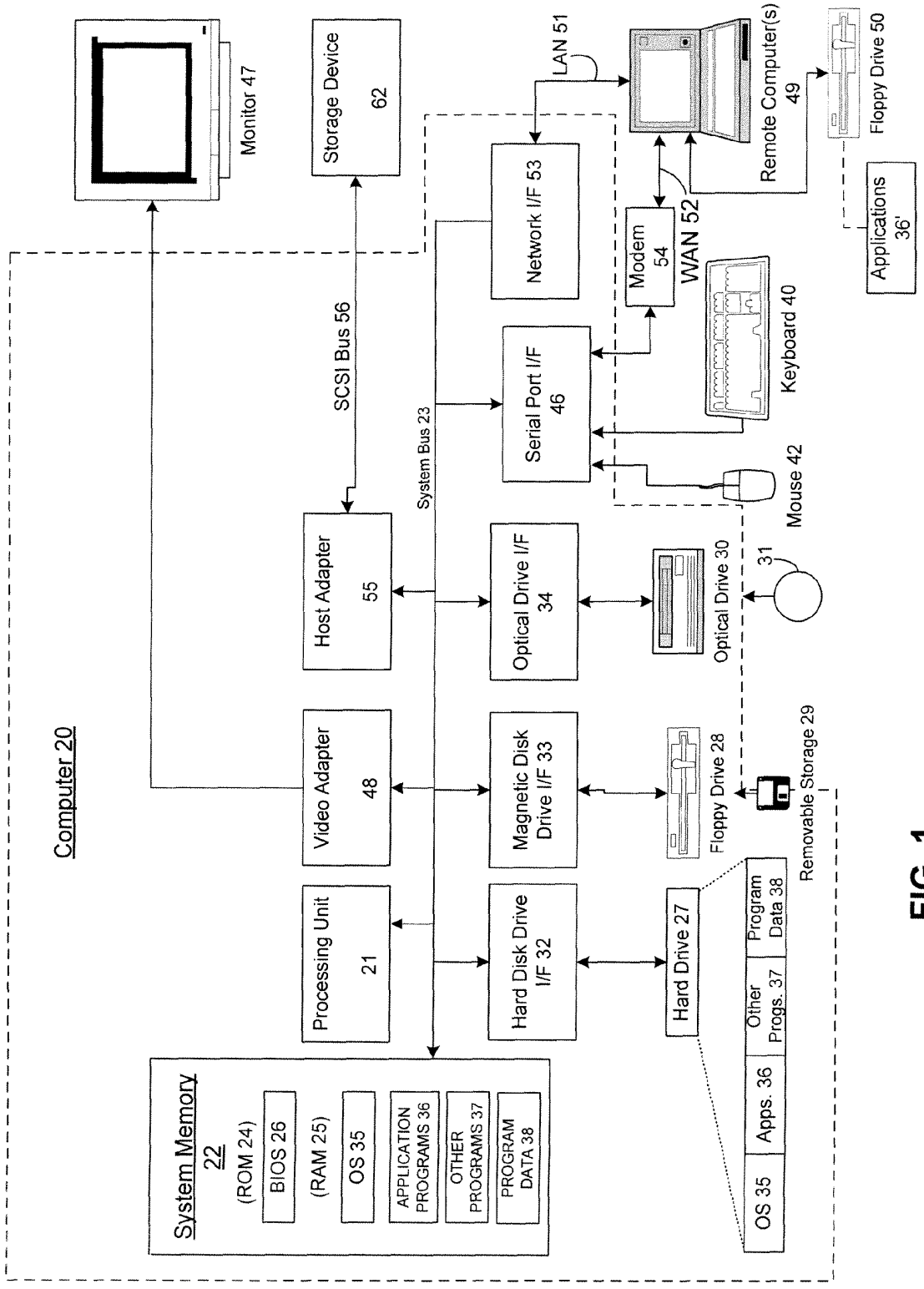

Computer 20
System Memory 22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
APPLICATION PROGRAMS 36
OTHER PROGRAMS 37
PROGRAM DATA 38
Processing Unit 21
Video Adapter 48
Host Adapter 55
Hard Disk Drive I/F 32
Magnetic Disk Drive I/F 33
Optical Drive I/F 34
Serial Port I/F 46
Network I/F 53
System Bus 23
SCSI Bus 56
Monitor 47
Storage Device 62
Hard Drive 27
Floppy Drive 28
Optical Drive 30
31
Removable Storage 29
OS 35
Apps. 36
Other Progs. 37
Program Data 38
Mouse 42
Keyboard 40
Modem 54
WAN 52
LAN 51
Remote Computer(s) 49
Floppy Drive 50
Applications 36'

FIG. 1

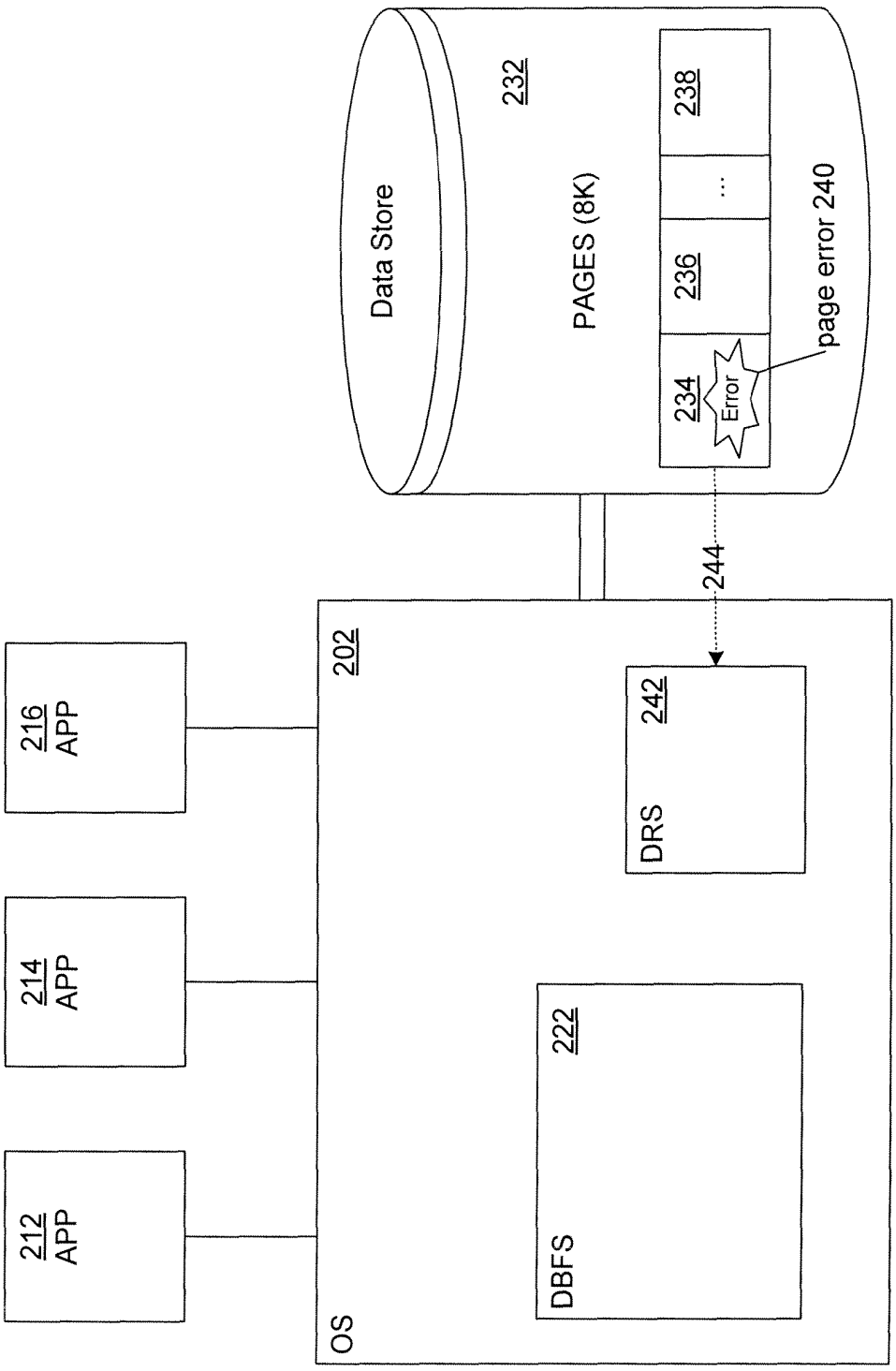

Data Store
232
PAGES (8K)
238
...
236
234
Error
page error 240
244
202
216
APP
214
APP
212
APP
DRS
242
DBFS
222
OS

FIG. 2

FIG. 3

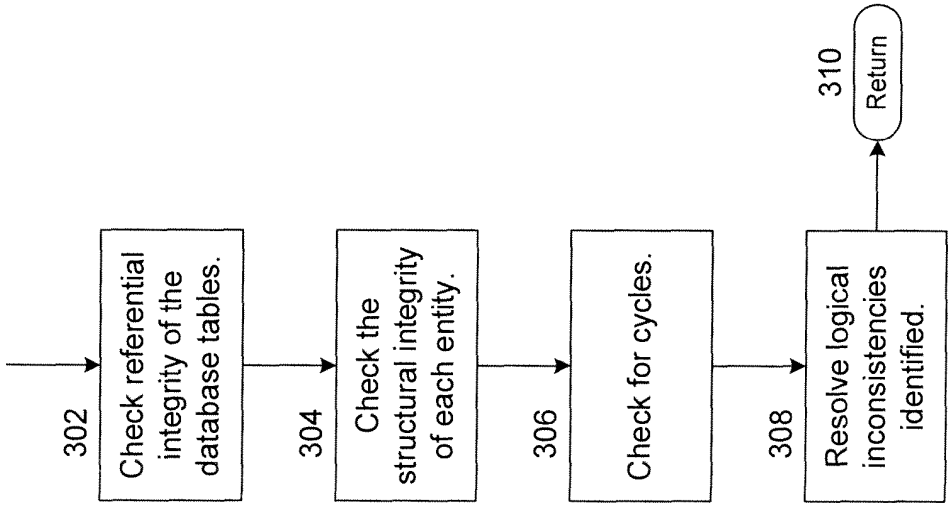

302
Check referential integrity of the database tables.
304
Check the structural integrity of each entity.
306
Check for cycles.
308
Resolve logical inconsistencies identified.
310
Return

FIG. 4

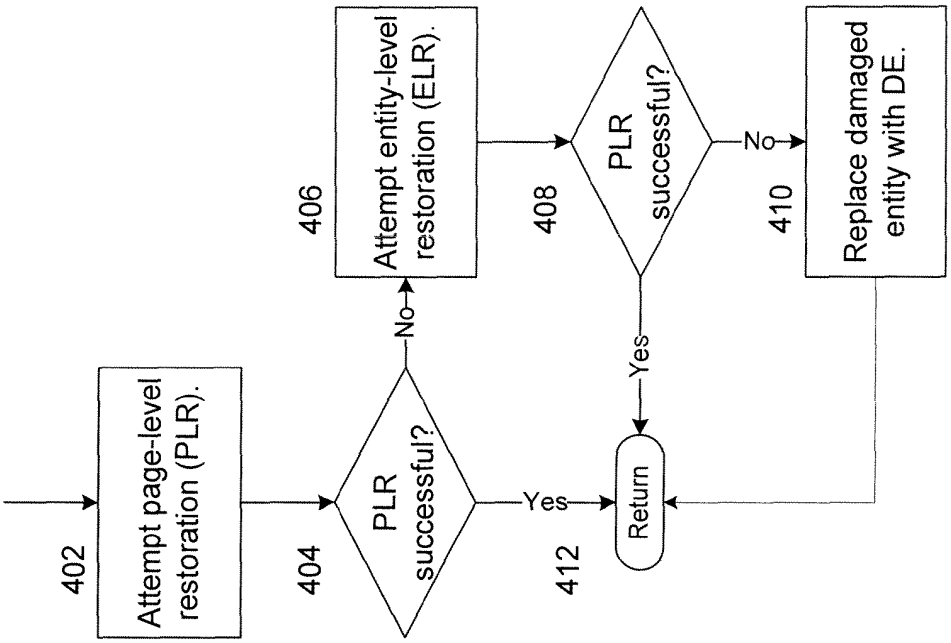

402
Attempt page-level restoration (PLR).
404
PLR successful?
No
406
Attempt entity-level restoration (ELR).
408
PLR successful?
No
410
Replace damaged entity with DE.
Yes
412
Return
Yes

FIG. 5B

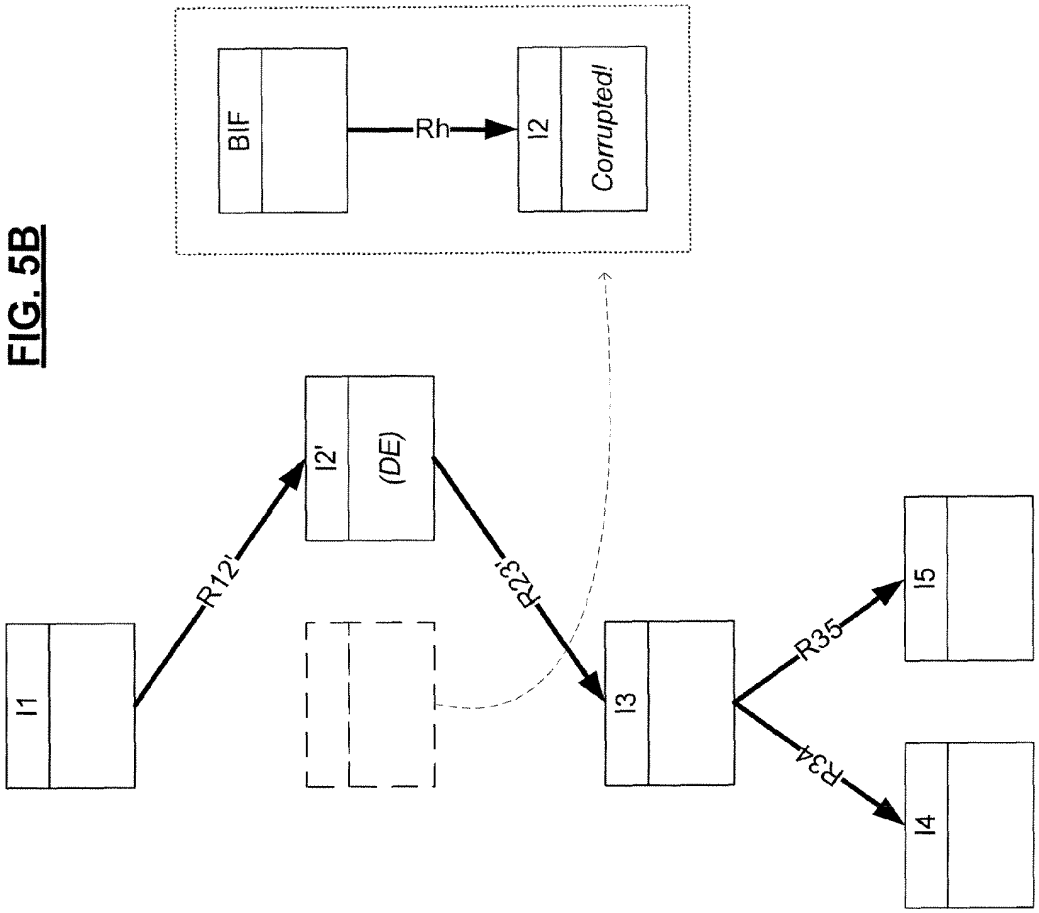

BIF
Rh
I2
Corrupted!
I1
R12'
I2'
(DE)
R23'
I3
R35
I5
R34
I4

FIG. 5A

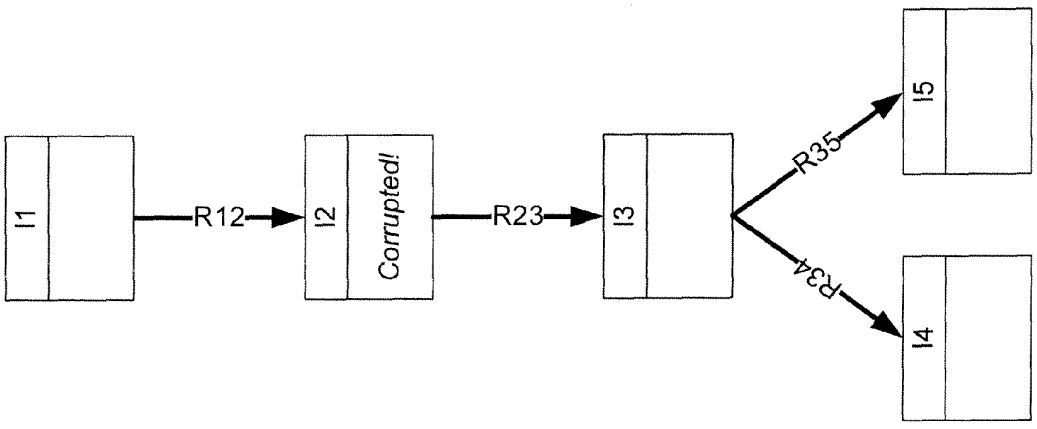

I1
R12
I2
Corrupted!
R23
I3
R35
I5
R34
I4

FIG. 6B

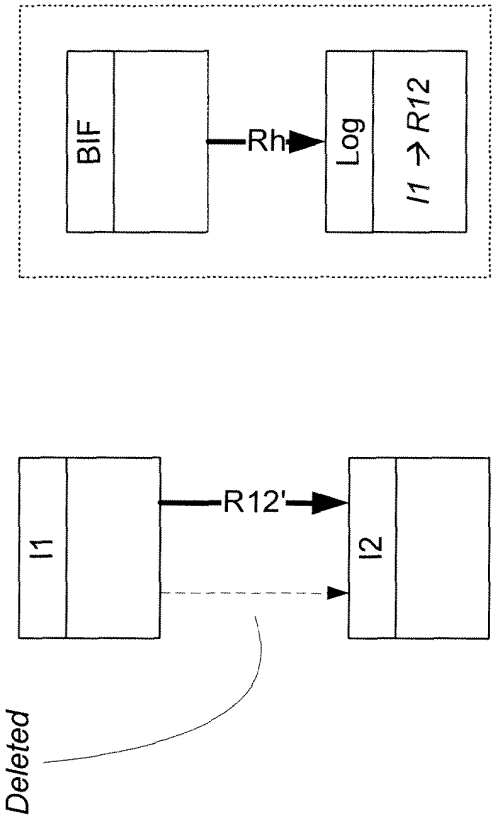


FIG. 6A